Some Thoughts on Forward Induction in Multi-Agent-Path Finding Under Destination Uncertainty

Bernhard Nebel

Albert-Ludwigs-Universität Freiburg, Institut für Informatik
nebel@uni-freiburg.de
http://www.informatik.uni-freiburg.de/~nebel

Abstract. While the notion of implicit coordination helps to design frameworks in which agents can cooperatively act with only minimal communication, it so far lacks exploiting observations made while executing a plan. In this note, we have a look at what can be done in order to overcome this shortcoming, at least in a specialized setting.

1 Introduction

In implicitly coordinated multi-agent path finding under destination uncertainty (MAPF/DU) [5] (and more generally in epistemic planning [3,2]), we have so far concentrated on generating plans in a way such that each agent tries to generate situations from which the other agents can provably find a plan that guarantees success. This means in particular that we do not make use of observations made during the execution of a plan in order to learn something about the destinations (or gain other information), something similar to what is called *forward induction* [1] in game theory or *plan recognition* [8,6] in the area of automated planning.

Not making use of observations implies that agents cannot use their actions in order to signal their intention. For these reasons, plans might be longer than necessary or an instance might not be solvable, although by making inferences about the intentions of the other players, the instance could be solvable. In this paper, we will analyze, in which situations one can make use of observations and how this can be integrated into the planning process.

In order to so, we will introduce the basic notation and terminology in the next section. In Section 3, we will then analyze how to modify our notion of a solution concept for MAPF/DU.

2 Background

The multi-agent path finding problem in its simplest form could be stated as follows. The environment is modelled as an undirected, simple graph $G = (V, E)$. A *configuration* of *agents* A on the graph G is an injective function $\alpha\colon A \to V$.

For $i \in A$ and $v \in V$, by $\alpha[i/v]$ we refer to the function that has the same values for all $j \neq i$ as α, but for i it has the value v: $\alpha[i/v](i) = v$.

Given a *movement action* of agent i from v to v' and a configuration α, a *successor configuration* $\alpha' = \alpha[i/v']$ is generated, provided $\alpha(i) = v$, $(\alpha(i), \alpha'(i)) \in E$, and there exists no j with $\alpha(j) = v'$. The *MAPF problem* is then to generate for a given *MAPF instance* $\mathcal{M} = \langle A, G, \alpha_0, \alpha_* \rangle$ with a given set of agents A, a given graph G, the *initial configuration* α_0, and the *goal configuration* α_*, a sequence of movements from α_0 to α_*. We always assume that such movement plans are *cycle-free*, i.e., that during the execution of such a plan no configuration is reached twice. We call a plan *successful* for a MAPF instance if it transforms α_0 into α_*. Since in the following we only consider successful movement plans, we just call them *plans*. If there exists such a plan for a given instance, we call the instance *solvable*.

For this basic version of the MAPF problem, most interesting questions concerning computational properties have been answered already in a paper by Kornhauser et al. in 1984 [4]. It is known that solvability can be decided in cubic time and the plan length and the time to find a plan is also bounded cubicly. Finally, solving the bounded planning problem (corresponding to the optimization problem) is NP-complete [7].

2.1 Generalized MAPF

Plans are usually generated in a centralized manner and the agents then follow the plan. In our generalized setting, we assume all agents plan by themselves and the goals of the agents are not common knowledge any longer. Instead only the agent itself knows its own destination. Common knowledge are the possible destinations for each agent, formalized by a *destination function* $\beta\colon A \to 2^V$, with the constraint that for all $i \in A$ either the real destination is among the possible ones, i.e., $\alpha_*(i) \in \beta(i)$, or $\beta(i) = \emptyset$, because agent i already arrived (and is not allowed to move anymore). We require further that all combinations of possible destinations are consistent, i.e., $\beta(i) \cap \beta(j) = \emptyset$ for all $i \neq j \in A$.

In the original MAPF problem, the state space for the planning process is simply the space of all configurations α of the agents in the graph. For the MAPF problem with destination uncertainty we also have to take into account the possible belief states of all the agents. For this reason, we have to make the possible destination function part of the state space as well, i.e., an *objective state* is now the tuple $s = (\alpha, \beta)$, which captures the *common knowledge* of all agents. Since the precise destinations are not common knowledge any longer, it is necessary to have some form of signal so that an agent can tell the other agents that it does not want to move any more—meaning it has reached its final destination. Only with such a *success announcement* the agents will in the end know that everyone has reached its destination.

An instance of the problem is now given by the tuple $\mathcal{M}_{DU} = \langle A, G, s_0, \alpha_* \rangle$, with the set of agents A, the graph $G = (V, E)$, the initial objective state $s_0 = (\alpha_0, \beta_0)$, and the goal configuration α_*. Movement actions change the configuration α, while success announcements change the destination function

β. If an agent i makes a success announcement while being in location v, we change the destination function to $\beta[i/\emptyset]$, signaling that the agent has reached its destination and is not allowed to move anymore. The goal state is reached if for all agents i, $\alpha(i) = \alpha_*(i)$ (the destination has been reached) and $\beta(i) = \emptyset$ (success has been announced).

2.2 Branching Plans

When an agent i is starting to generate a plan, the agent knows, of course, its true destination $\alpha_*(i)$. The subjective view of the world is captured by the tuple $(\alpha, \beta, i, \alpha_*(i))$, which we call *subjective state of agent i*. Given a subjective state $(\alpha, \beta, i, \alpha_*(i))$, we call (α, β) the *corresponding objective state*. Using its subjective state, agent i can plan to make movements that eventually will lead to a goal state. Most probably, it will be necessary to plan for other agents to move out of the way or to move to their destination. So, the planning agent has to put itself into the shoes of another agent j: i must make a *perspective shift* taking j's view. Since i does not know the true destination of j, i must take all possibilities into account and plan for all of them. In other words, i must plan for j using all possible subjective states of j: $s_v^j = (\alpha, \beta, j, v)$ for $v \in \beta(j)$. When planning for each possible destination of j, the planning agent i must pretend not to know the true destination of itself because it plans with the knowledge of agent j, which is uncertain about i's destinations.

All in all, a plan in the context of MAPF with destination uncertainty is no longer a linear sequence, but a *branching plan*. Furthermore, it is not enough to reach the true goal state, but the plan has to be successful for all possible destinations of all the agents (except for the starting agent i, who knows its own destination).

Such a branching plan corresponds roughly to what has been termed *policy* in the more general context of *implicitly coordinated epistemic planing* [2,3]. In order to illustrate the concept of a branching plan, let us consider the example in Figure 1. Here *square agent S* knows that its destination is v_3 (the solid square)

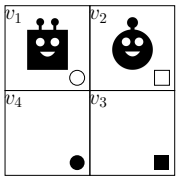

Fig. 1. Small example with square agent (S) and circle agent (C)

and the *circle agent C* knows that its destination is v_4 (the solid circle). However both are unsure about the destinations of the other agent. So S knows that v_1 and v_4 are possible destinations for C. C in turns knows that v_2 and v_3 are possible destinations for S.

Let us now assume that *square agent* S moves first to v_4. Now S puts itself into the shoes of *circle agent* C and reasons about what C would do, if v_1 is C's destination, and how C would continue if v_4 is C's destination. In the former case, C moves to v_1 and announces that it has reached its destination. In the other case, it will also move to v_1, offering S the possibility to move to its destination, whether it is v_2 or v_3. After that, C could move to its destination. All in all, a branching plan could look as depicted in Figure 2. In this plan,

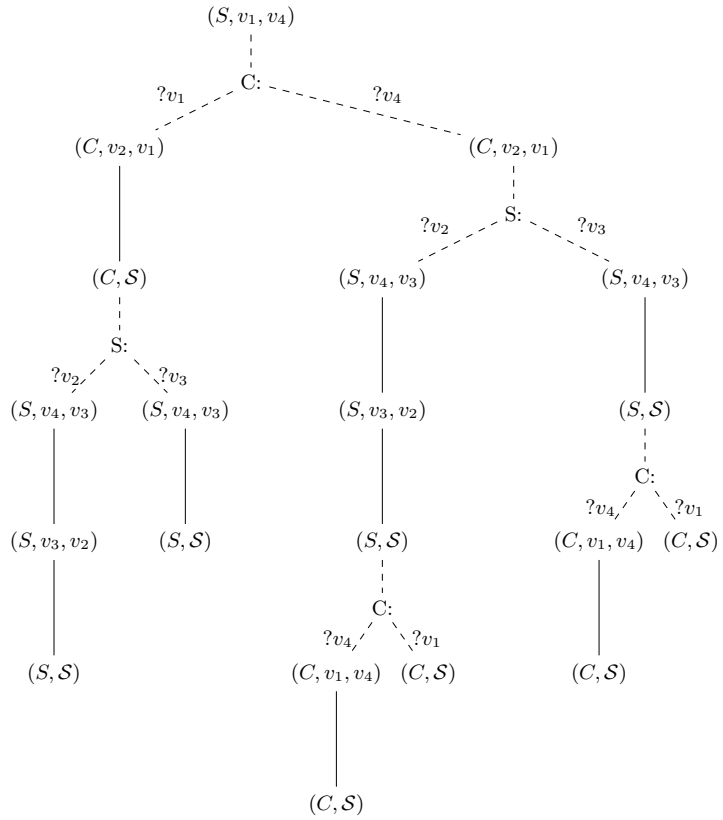

Fig. 2. Branching plan

each perspective shift to another agent is followed by branching according to the possible destinations of the agent. In general, we do not always require such branching because the agent might decide to move independently of its own destinations. One of the main results is then that all successful branching plans need to branch only on so-called *stepping stone* situations [5, Theorem 5]. These are configurations in which one agent has unblocked ways to all its possible destinations, and for those destinations, there are successful subplans after that

agent has reached it. For example, in Figure 2, after S's initial movement from v_1 to v_4, there is neither a stepping stone situation for C not for S. However, after C had then moved from v_2 to v_1, C had created a stepping stone situation for S. S can move now uninterrupted to v_2, announce success and there is a successful plan afterwards for C. In case v_3 is the real destination, S can move there and again there is a successful plan afterwards for C. From the fact that a plan needs only to branch on stepping stones, it follows that these branching plans need to have only polynomial depth.

2.3 Joint Execution Model

After all agents have planned, we have a *family of plans* $(\pi_i)_{i \in A}$. *Joint execution* of this family of plans is then performed in an asynchronous, interleaved fashion. From all the agents i that have as their first action one of their own moves, one agent is chosen and its movement is executed. This is very similar to what happens in real-time board games, such as *Magic Maze*. The player who acts fastest carries out the action. For all the other agents the following happens: Either the movement was anticipated and then the movement is removed from the plan or the agent has to replan from the new situation. The interesting question is, whether such an asynchronous, distributed execution is guaranteed to eventually lead to the desired goal configuration and how many steps it takes to reach the common goal.

3 Exploiting Observations while Executing

As has been shown, under some reasonable conditions it is possible to guarantee success, provided that there is at least one agent which is able to come up with a plan initially [5]. However, there are also situations which look easily solvable, but it turns out that our notion of *implicit coordination* does not capture this. One such example is shown in Figure 3. The square agent wants to go v_4 and knows that the circle agent wants to go either to v_2 or to v_5. Similarly, the circle agent wants to go to v_2 and knows that the square agent wants to go either to v_1 or to v_4.

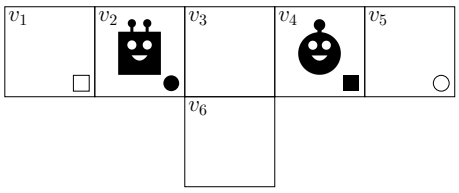

Fig. 3. MAPF/DU instance that is only solvable using inferences about observations

If the square agent tries to solve the instance, it will try to create a *stepping stone* situation [5, Sect. 3.2] for the other agent. The only possible way to do

that appears to be to move to v_6. Now the circle agent can move to both possible destinations. Unfortunately, after moving to v_2, the circle agent cannot any longer guarantee that the square agent can reach both of its possible destinations. Specifically, the square agent is blocked from reaching v_1. So, the move of the square agent to v_6 does not create a stepping stone for the circle agent. As is easy to see, all other movements of the square agents do not create a stepping stone either. Since the situation is completely symmetric, also the circle agent is unable to create a stepping stone.

If we try to explain movements by assuming that the other agents act rationally, we may assume that they always try to come up with shortest possible plans. Therefore, if the circle agents observes the square agent moving to v_6, the circle agent may rightly conclude that v_1 cannot be the actual destination of the square agent. Because if it were, then the square agent would have moved there directly announcing success, which would have led to an overall shorter plan. So, for the remaining part of the plan both agents can assume that v_1 is not the actual destination of the square agent. This implies that the circle agent can safely move to either v_2 or v_5 and afterwards the square agent can move to the only possible destination, namely v_4, which then solves the previously unsolvable instance.

Below we will discuss how to generalize this kind of reasoning.

3.1 Safe Abduction

Abduction is the inference to the best explanation, given an observation and a background theory. This is apparently what we are using when drawing the above conclusion that v_1 is not the destination of the square agent. In general, abduction is an "unsafe" inference in that the best explanation is not necessarily the correct one. For instance, often the best explanation for the malfunctioning of a device is based on a single-failure assumption, which might nevertheless not be the right explanation.

In our context, incorrect explanations could easily lead to situations, where destinations are no longer accessible, turning a solvable instance into an unsolvable one. In order to avoid that we will only accept explanations that are safe in the sense that they do not exclude a destination that is still be possible. Using the criterion of aiming for shortest plans, it may, however, still be possible to infer meaningful information.

3.2 Observations and Explanations

So what should count as an *observation* that needs an explanation? As in the example above, a meaningful observation is a sequence of movements by one agent i starting at a node v ending at node v' without interruptions by other agents. In the example, this would be the movements of the square agent from v_2 over v_3 to v_6. In this example, one might also could consider the movement from v_2 to v_3 as one observation.

In order to explain an observation, we take all possible destinations of the moving agent i into account and generate shortest plans for each of these destinations $v^*_{i,1}, \ldots, v^*_{i,k}$, starting with movements of agent i at node v not using the prefix from v to v'. Call these plans $\pi_{i,j}$. In creating these plans one has to take into account that the other agents do not know the destinations of that agent. Similarly, create shortest plans that include the prefix from v to v' and call these plans $\pi'_{i,j}$. Note that all these sub-plans may also use safe abduction!

Assuming that $|\pi|$ denotes the execution cost of plan π, we now compare the plans for all destinations $v_{i,j}$. If $|\pi_{i,j}| < |\pi'_{i,,j}|$,[1] we conclude that agent i cannot rationally try to reach destination $v^*_{i,j}$ moving from v to v'. In fact, all agents observing this behavior can conclude this and agent i (and everybody else) is aware that everybody else knows that. In other words, after agent i moved from v to v', it is common knowledge that agent i is trying to reach one of the destinations $v_{i,j}$ such that $|\pi'_{i,j}| \leq |\pi_{i,,j}|$. Note that we included all destinations $v^*_{i,j}$ with $|\pi'_{i,j}| = |\pi_{i,,j}|$, since there is no reason do dismiss $v^*_{i,j}$ after having reached v'.

One important prerequisite for this kind of inference to be correct is, however, that agents indeed always generate a shortest plan. And this does not only concern the overall plan, where we measure the length as the longest trace through the branching plan. Instead, this should be true also for each sub-plan at each point, where a perspective shift happens. This is something we currently do not require from our plans when giving success guarantees. Furthermore, we explicitly do not require to branch at each point where a perspective shift happens. However, it is, of course, possible to make that a requirement.

The most interesting feature of using this kind of inference is that it also can make instances solvable that were unsolvable before. For the example in Figure 3, we showed that no stepping stone exists, so that it cannot be solved by an implicitly coordinated branching plan. However, using the notion of safe abduction, C can come up with the plan of moving to v_6. Now this is definitely not a prefix of an optimal plan for solving this instance when v_5 is C's destination. On the other hand, there exists no plan at all to solve the instance when v_2 is the destination, i.e., plan length is infinite. In other words, everybody can safely assume that C does not have v_5 as a destination. Using this assumption, the instance can then be easily solved.

However, it turns out that sometimes the agent might not have the right option to act in order to signal that a possible destination can be excluded. Let us reconsider the example from above but place C initially into cell v_6. Now the problem is that the only way to signal that v_5 is not C's destination is to do nothing. However, doing nothing cannot be observed in our asynchronous execution model. The way out here could be to introduce an observable *wait* action, which induces also execution costs. Then no successful shortest sub-plan for a particular destination could contain this action. On the other hand, if there does not exist a plan for a possible destination where the agent moves first, a wait action does not matter, because plan length is infinite in any case.

[1] If no plan can be found, then we assume infinitely large execution costs.

In our modified example, where C is initially in cell v_6, there exists a plan for C's destination v_5 with C moving first, hence a wait action could not be part of a shortest plan. For C's destination v_2 on the other hand, there does not exist any plan with C moving first. So, a wait action is appropriate here. So, a wait action can signal that only those destinations remain for which there exists no plan initially; in our example, this would mean that v_2 must be the actual destination.

Actually, an alternative to a wait action might be to make one move and then return to the original location. This would also signal that only those destinations are possible for which there is no initial plan where the agent moves first.

In any case, regardless of whether we use a *wait* action or a back and forth movement, we seem to violate the requirement set out earlier, namely that plans should by cycle-free, which in the case of branching plans translates into the requirement that no objective state should be visited twice on a possible execution trace. However, the movement is made in order to change the common knowledge (see below) and in so far, no cycle is created.

3.3 Forward Induction

During the MAPF/DU planning process, common knowledge over all possible destinations is maintained using the possible destination function β, i.e., $\beta(i)$ is the commonly known set of all possible destinations for agent i. This set is reduced to the empty set whenever agent i makes a success announcement. If we use safe abduction as described above, we can reduce the set of possible destinations $\beta(i)$ to all those that are still possible according to the definition in the previous subsection. Interestingly, since this is common knowledge, this reduction can be propagated to the entire sub-plan following i's movement (or inaction).

In order to illustrate that this can even proceed over more than one stage, let us consider a more complex example, where we add a third agent T, the triangle agent (Figure 4).

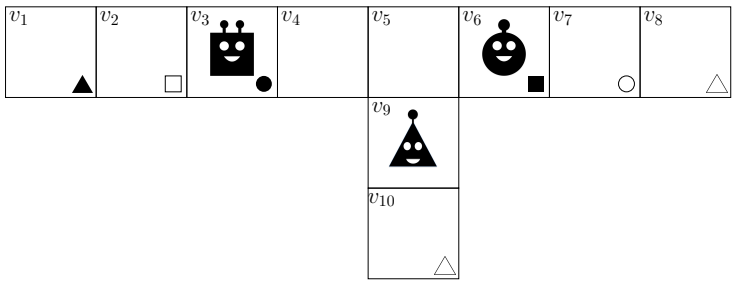

Fig. 4. More complex example

Here, T could start by moving to v_4, signalling that v_{10} is not its destination. The only way for C would be to move to v_{10} (in order to help T later on), in

order to allow T to move to its possible destination. Not that at this point we are not entitled to make the inference that v_7 is not the destination of C, because C could not have moved there announcing success with the guarantee that the remaining problem could be solved. If T's goal were v_8, it could move there and the rest would be easy for C and S. However, instead it moves to v_9 signalling that v_8 is not its destination. Now it is common knowledge that v_1 is T's destination. However C and S do not have knowledge about their respective destinations. S would unblock the way for T by moving to v_6, and T could move to v_1, announcing success. C cannot make any meaningful move, so S has to move, either to v_2 or v_3, in order to allow for C either to move to its destination v_7, or to signal that this not C's destination. Now C could execute a wait action in order to signal that v_7 is not C's destination. So, S could happily move to v_6 and announce success, after which C could easily finish.

3.4 Computational Complexity

From a computational complexity point, most probably nothing changes. The construction in the proof of Theorem 11 of the original paper [5] still works. In particular, forward induction is of no help and not a hindrance in deciding the constructed MAPF/DU instance. For proving PSPACE membership, one has to prove a generalized stepping stone theorem. A generalized stepping stone is now a state such that either the agent can reach all possible destinations, announce success and the solvability of the simplified problem can be guaranteed (as usual), or the movements of the agent end in a state such that we can safely abduct and can at least eliminate one possible destination. With that, polynomial depth of the branching plan follows and then one could easily guess and check all traces iteratively. One has to guess also the depth of plans (which subsequently have to be verified) in order to allow for the verification of safe abductions.

4 Outlook

Although the complexity probably does not change, algorithmically things become more involved. In fact, one might want to consider only special cases of safe abduction inferences in order to reduce computational overhead. For example, one might only consider situations when success announcements are possible and ignored, as in the above examples. Otherwise the computational burden might be too high. In particular, it remains unclear whether we could reduce the worst case execution costs, which are what we are interested in when proving success guarantees.

An interesting question then comes up related to the omniscience problem. If an agent does something that another agent can take as a signal, then the other agent actually has to recognize that as a signal, otherwise the plan of the acting agent might not work out. In other words, all agents have to use the same level of reasoning.

All in all, the ideas spelled out here might hopefully serve as a starting point for defining a notion of implicit coordination that also takes into account the observation about actions of the other agents. Hopefully, this might also lead to generalizing these ideas to more general settings such as epistemic planning with monotonic uncertainty reduction or even general epistemic planning.

References

1. Battigalli, P., Siniscalchi, M.M.: Strong belief and forward induction reasoning. J. Economic Theory **106**(2), 356–391 (2002). https://doi.org/10.1006/jeth.2001.2942
2. Bolander, T., Engesser, T., Mattmüller, R., Nebel, B.: Better eager than lazy? How agent types impact the successfulness of implicit coordination. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference (KR-2018). pp. 445–453 (2018), `https://aaai.org/ocs/index.php/KR/KR18/paper/view/18070`
3. Engesser, T., Bolander, T., Mattmüller, R., Nebel, B.: Cooperative epistemic multi-agent planning for implicit coordination. In: Proceedings of the Ninth Workshop on Methods for Modalities (M4MICLA-17). pp. 75–90 (2017). https://doi.org/10.4204/EPTCS.243.6
4. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: 25th Annual Symposium on Foundations of Computer Science (FOCS-84). pp. 241–250 (1984). https://doi.org/10.1109/SFCS.1984.715921
5. Nebel, B., Bolander, T., Engesser, T., Mattmüller, R.: Implicitly coordinated multi-agent path finding under destination uncertainty: Success guarantees and computational complexity. J. Artif. Intell. Res. **64**, 497–527 (2019). https://doi.org/10.1613/jair.1.11376
6. Ramírez, M., Geffner, H.: Plan recognition as planning. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009). pp. 1778–1783 (2009), `http://ijcai.org/Proceedings/09/Papers/296.pdf`
7. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N × N extension of the 15-puzzle is intractable. In: Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86). pp. 168–172 (1986), `http://www.aaai.org/Library/AAAI/1986/aaai86-027.php`
8. Schmidt, C.F., Sridharan, N.S., Goodson, J.L.: The plan recognition problem: An intersection of psychology and artificial intelligence. Artif. Intell. **11**(1-2), 45–83 (1978). https://doi.org/10.1016/0004-3702(78)90012-7